Eötvös Loránd Tudományegyetem

Informatikai kar

Információs Rendszerek Tanszék

SQL lekérdezés újraírása, mintázatok és több információ felfedésének céljából

dr. habil. Kiss Attila

tanszékvezető, egyetemi docens

Dénes Benjámin

programtervező informatikus BSc

Budapest, 2019

Köszönöm dr. Kiss Attila tanár úrnak, hogy megmutatta nekünk az adattudomány izgalmas világát és végig segített a dolgozat elkészítésében.

Tartalomjegyzék

Bevezetés

Az adattudomány az informatika tudományok (egyik) legdinamikusabban fejlődő területe. Felhasználási spektruma az informatikáéval azonos. Ahol adatok keletkeznek bármilyen formában, ott felfedezhetünk valamilyen összefüggést, rendszert. Ezeket optimalizációra, felfedezésre, ismereteink bővítésére használjuk.

Elektromos eszközeink segítségével adatokat szinte életünk minden másodpercében létrehozunk, így mindenki számára érdekes és hasznos lehet ez a terület.

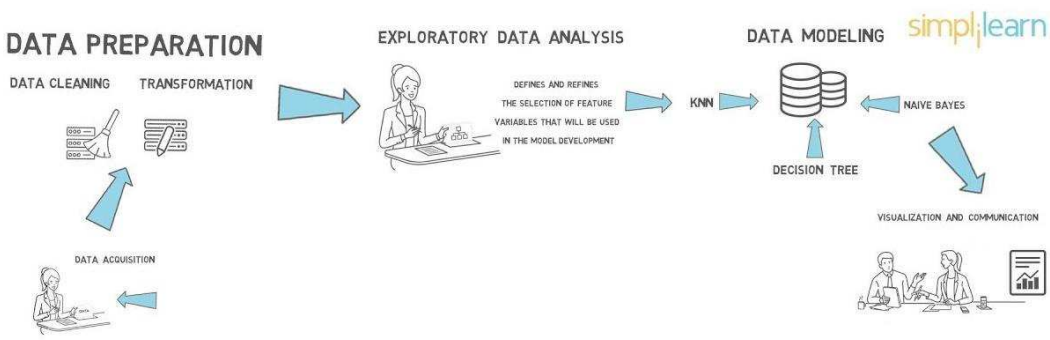
Az átlag ember számára életmódunk egyszerűsítését, teendőink hatékonyabbá és gyorsabbá tételét jelenti, magasabb profithoz jutást, vagy kiadásaink csökkentését.

Az adattudomány legjelentősebb algoritmusai mára a mesterségei intelligencia alapjait fektették le.

Ebben a dolgozatban egy egyszerű algoritmust szeretnék bemutatni, melynek célja adatok és minták felfedezése SQL környezetben. Az algoritmust 4 kutató publikálta „Data Exploration with SQL using Machine Learning Techniques” címmel, nagy mennyiségű csillagászati adathalmazon tesztelve.

Dolgozatom célja az algoritmus implementálása saját értelmezésemben és ezsközeimmel, illetve eköré egy kényelmesen használható Python és SQL alapú grafikus környezet készítése.

Kulcsszavak: data exploration, query rewriting, decision trees, big data



Motiváció: a nagyvilág

Big data-nak nevezzük azt a mennyiségű adatot, amit a hagyományos adatbáziskezelő rendszerekkel már nem tudunk feldolgozni. A világ teljes adatmennyisége exponenciálisan növekszik. Ez azt jelenti, hogy 90%-a csak az elmúlt néhány évben keletkezett. [2] Néhány példa:

Az LSST teleszkóp célja, hogy éjszakánként a teljes égboltot fotózza, 15 terabájtot is képes lesz rögzíteni. [3] A 2 milliárd euróból épülő SKA teleszkópot úgy becsülik, legalább 160 TB adatot fog rögzíteni másodpercenként. [4]

2019-ben elkészült az első kép egy fekete lyukról 5 petabájtnyi adatból. [5]

A technikai kapacitások szinte hihetetlenek, a problémát már nem a tárolás vagy elérés jelenti. Inkább az, hogy nem tudjuk mit és hogyan keressünk a rengeteg adat közt, hogy értékes információt nyerhessünk ki.

Az OpenAI cég Dota 2 nevű játékhoz készített mesterséges intelligenciája egyetlen nap alatt legalább 180 évnek megfelelő játékinformációt képes feldolgozni. Megerősítéses tanulással fejlődik és 2019-ben sikerült megvernie a regnáló világbajnok csapatot. [6]

A Facebook oldalán naponta másfél milliárd felhasználó fordult meg, a milliós adathalmazok percenként keletkeznek. [7]

De elég csak egy nagyvárosi supermarket vásárlási adataira gondolnunk. Egy egyszerű online szolgáltatásra, webshopra vagy játékra, amiknek több millió felhasználója van az egész világon.

Az ipari realitás eközben, hogy a világ legnagyobb vállalatai petabájtnyi hasznos adattal rendelkeznek, melyet egyáltalán nem vagy alig dolgoznak fel. Ezen adatokból nyert információk hosszú távon talán milliárdos extra profitot hoznának.

Jól látható tehát, hogy rendkívül hasznos tanulmányoznunk a legegyszerűbb adatokban rejlő tudást.

Visszatérés a dolgozatra

Ennek a világnak egy kis szeletét szeretném bemutatni, egy nagyon egyszerű felfedező (data exploration) algoritmust.

A feladat

- **Adott egy SQL adatbázis**, melyben hasznos adatokra leltünk **egy lekérdezés** segítségével.

- Arra vagyunk kíváncsiak, **követnek-e** valamilyen **mintát** az adatok, azon kívül, amit a lekérdezés alapján már felfedeztünk vagy felfedezni véltünk.

Más szavakkal, **át lehet-e írni** a lekérdezésünk **más feltételekkel**, amitől esetleg egyszerűbb, rövidebb, vagy információdúsabb lesz. Az oszlopok számától függően valószínüleg nem is mindegyik lesz érdekes számunkra.

Ha megpróbálunk akár kézzel keresni egy ilyen átírást, rögtön felmerül a kérdés, hogy a válasz soroknak mennyire kell lefedni az eredeti lekérdezés válaszait.

- Az nem gond, sőt **előnyös** is, ha **egy bővebb válasz halmaz**t kapunk. Ez a felfedezés második része. Az eredeti válaszokat pedig lehetőleg mindet tartsuk meg, hogy pontosabb legyen az egyezés az eredeti lekérdezéssel.

Fejlesztői dokumentáció

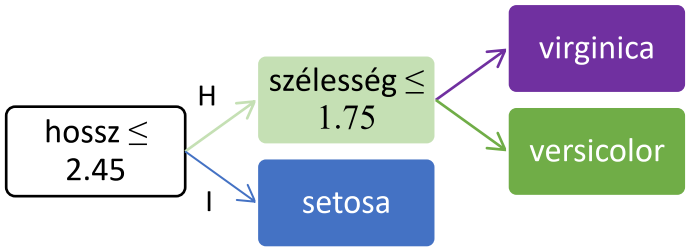
Az algoritmus ötlete

- Tehát szeretnénk átírni a lekérdezés feltételeit, de hogyan? Egy adatbányászatban gyakran használt módszert használunk, **döntési fát építünk**. (Decision tree learning)

- A döntési fához szükségünk van **adatsorok**ra és azok **osztályozására**, amik alapján a fa csúcsai felépülnek. Egy csúcsba egy feltétel kerül, ami ha teljesül, az adott osztályra mutat, ha nem teljesül, akkor egy másikra. Végül a levélbe kerül a szülő feltételek konjunkciójával kapott osztály.

Egy példa: írisz (nőszirom) virágok osztályozása a szirom alapján

hossz	szélesség	faj
1.9	0.4	setosa
3.0	1.1	versicolor
4.1	1.3	versicolor
6.0	2.5	virginica



Egy nagyobb, részletesebb ábra megtalálható a Scikit-learn dokumentációban. [8]

- Az osztályzást a lehető legegyszerűbben készítjük el. Az adatbázis azon sorai, amiket a **válaszban szeretnénk látni, a pozitív osztályba** kerülnek, amiket pedig nem, a **negatív** osztályba.

- A pozitívak meghatározásához elég csak futtatni a lekérdezést, de mik legyenek a negatív példák? Ha a komplementer adathalmazt választjuk, akkor nem tudunk újabb adatsorokat felfedezni, amit a feladatban második célnak tűztünk ki.

- Tehát definiáljuk egy **lekérdezés negáltját**, ami lényegében a feltételek negálását jelenti, de nem mindét. Leglább 1 feltétel legyen negálva, a többi szerepelhet negálva, nem negálva, vagy el is hagyhatjuk. Például „A és B és C” egy negáltja „nem B és C”.

- A lekérdezés negáltjai közül keressük azt, amelyik mérete (válasz sorok száma) a **legközelebbi az eredeti** lekérdezés **méretéhez**. Ezt nevezzük optimális negáltnak.

A teljes algoritmust pszeudókóddal itt nem részletezem, mert már megtalálható az említett cikkben [1] és a forráskódban is. Viszont a cikk egy lényeges része, hogyan keressük meg az optimális negált lekérdezést big data-ban.

Big Data

Ha a lekérdezésben N db feltételünk van, mindegyik 3-féleképpen szerepelhet, de le kell vonni, amikben egy sincs negálva. Vagyis a negált lekérdezések halmaza $3^N - 2^N$ db, **exponenciális**.

Hogy elkerüljük az adatbázis sokszori lekérdezését, inkább heurisztikus **becslés**t adunk minden negált lekérdezés méretére. Ehhez a feltételeket egyesével lekérdezzük és a méretük alapján a következő becslést adjuk: $|lekérdezés| = |A| \cdot \prod_f \tilde{f} = |A| \cdot \prod_f \frac{|f|}{|A|}$, ahol f egy feltétel a 3 lehetőség egyikében, A a teljes adathalmaz.

Hogy melyik a 3 közül, a következő problémánk. Adott N-1 db számhármasunk (x, 1-x, 1) és egy cél szám. Keressük az N-1 hosszú számsort, aminek elemei a számhármas egyik tagja és összegük a legközelebbi a cél számhoz.

Ez a „subset product problem” egy különleges esete, ami **NP-teljes**. Ha vesszük $\tilde{f}$ logaritmusát és felszorozzuk (kellően nagy) egész értékre, akkor „**subset-sum**” vagy „knapsack” problémát kapunk, ami kényelmesen megoldható (szorzásnál a nagy számok problémásak).

A lekérdezés

Az eddig megadott algoritmus akkor működik, ha a feltételek közt **csak ÉS kapcsolat** van. Nem lehetnek egymásba ágyazott kapcsolatok sem. (any, all, exists)

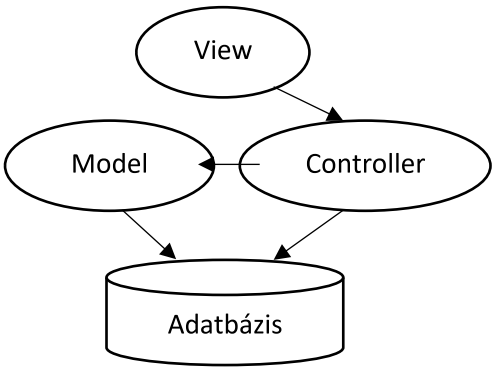
Egyébként egyszerű **SELECT-FROM-WHERE** lekérdezést várunk.

Táblákat lehet összekapcsolni és kereszt szorozni is. Utóbbi esetén, ha külső kulcsok alapján szűrünk, azon feltételeket egy külön sorban kell megadni a programban. Ezt **fix feltétel**nek nevezzük, mert nem kerülhetnek negálásra vagy hagyhatók el.

Forráskód

A kódot az Anaconda [9] (Python 3) programcsomaggal készítettem. Ebben megtalálható minden Python modul, de néhány nincs alapból telepítve. (Pár percet vesz igénybe.) Ezek a „graphviz”, „graphviz-python”, „mysqlclient”, „sqlparse”.

A program rétegei:



Model

Létrehoztam a Query osztályt, attribútumai:

+ _select: SELECT utáni attribútum lista

+ _from: FROM után megadott kifejezés

+ _joinon: fix feltétel, elnevezése a külső kulcs kapcsolatok miatt lett ez

+ _where: WHERE után megadott feltételek listája, „ and ” | „ AND ” mentén split-elve

Metódusok:

+ Run(): visszaadja egy Pandas.DataFrame-ben a lekérdezés eredményét

+ negate(): visszaadja egy listában a lekérdezés összes negáltját (következő metódushoz)

+ BruteForce(): visszaadja az optimális negáltat (ha több van, az összeset) és a méretét, miután futtatta az összes meghatározott negáltat (összehasonlításképp használhatjuk)

+ BalancedNegation(): visszaadja az optimális negáltat (ha több van, a legrövidebbet) és annak méretét, az előző oldalon bemutatott heurisztikus algoritmus

A SubSetSum(posW, negW, tW, preds) segédfüggvény az osztály mellett található. Dinamikus programozás módszerrel a preds feltétellista posW és negW sorrendben nem negált és negált esetekben becsült méretei alapján visszaadja a tW cél számhoz legközelebbi méretű feltétellistát és a méretét.

+ BuildLearningSet(negQ, name=”learn”): visszaadja a döntési fa építéséhez szükséges „+” és a negQ negált „-” osztályzattal ellátott példa adatsorokat Pandas.Dataframe-ben. Emellett létrehozza a „name.data” nevű fájlt, amibe szintén beleírja, illetve a „name.names” fájlt, amibe az oszlopok fejléce és típusa kerül. A két fájl a **C4.5** (újabb

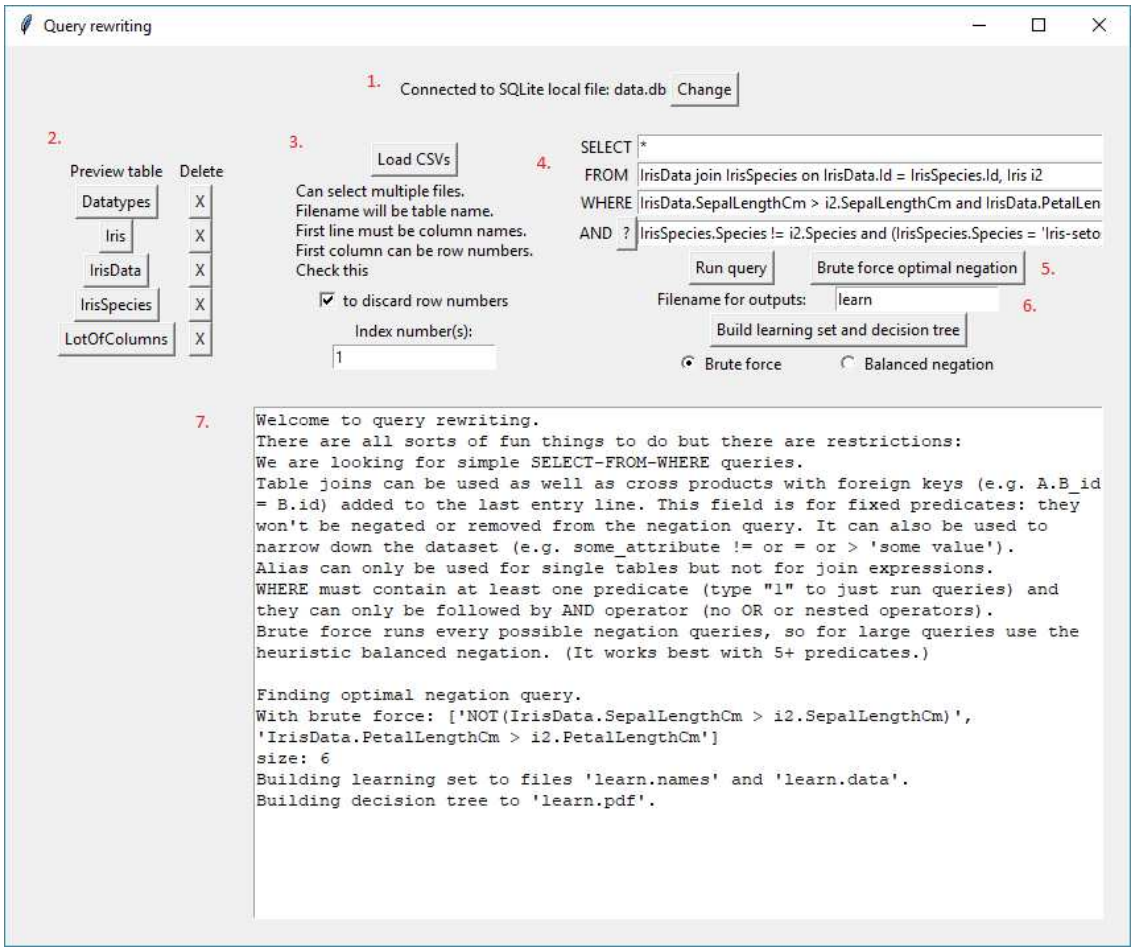
verzióban C5.0) néven ismert döntési fa építő algoritmus hivatalos implementációjában használható. [10]

+ DecisionTree(learn, name="learn"): felépít egy döntési fát, a Scikit-learn könyvtár DecisionTreeClassifier osztály algoritmusával, és exportálja a „name.pdf" fájlba. Ez az algoritmus más, mint a C4.5 és az implementáció kevésbé hatékony, mert csak számértékeket tud kezelni, így string (vagy dátum stb.) értékek helyett a sorszámukat jelenítem meg. A sorszámokhoz tartozó értékeket a „name-factorize.txt" fájlba is kiírja.

Az összes függvény és érdekes rész kommentálva van a kódban is.

Adatbázis + GUI

Ezután elkészítettem a program grafikus felületét TkInter, és az adatbázis funkciókat SQLAlchemy könyvtárakkal. A következő funkciók elérhetők:



View: frameConnect

Controller: buttonChangeCon(), buttonConnect()

1. Kapcsolódás adatbázishoz:

a) Indításkor a program könyvtárában létrehozott „data.db” nevű SQLite adatfájlhoz kapcsolódik. Ezt tetszés szerint átnevezhetjük és újakat hozhatunk létre.

b) Kapcsolódhatunk másik adatfájlhoz vagy adatbázishoz. Az SQLALchemy absztrakciós szintje lehetővé teszi 18 különböző DBAPI-hoz kapcsolódást. Kezdetnek csak az SQLite és MySQL opciókat vettem fel, de egyszerűen bővíthető.

V: frameList

C: ListTables(), AddTableButtons(name), buttonPreview(name), buttonDelete(name, b1, b2)

2. Táblák listája, előnézet, törlés: kapcsolódás után látjuk az adatbázis tábláinak listáját, ahonnan megnézhetjük külön ablakban az első és utolsó 30 sorát és törölhetjük a táblát

V: frameLoad

C: buttonLoadCSV()

3. Új tábla betöltése CSV fájlból: tetszőleges számú CSV fájlt tölthetünk be. Fájlnév lesz a tábla neve, az első sor az oszlopok fejléce, a többi sor lehet sorszámozott vagy nem.

V: frameQuery, frameOp

C: buttonRunQuery(), buttonBruteForce(), butttonBuild()

4. Lekérdezés futtatása: külön ablakban írja ki a választ.

5. Optimális negált meghatározása az összes lehetőség futtatásával.

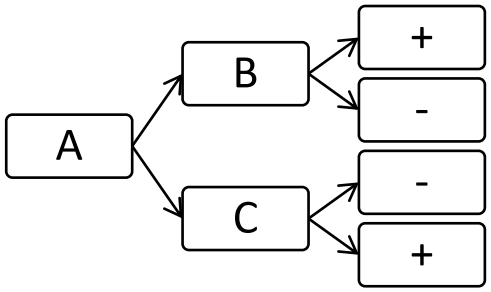
6. Döntési fa és a hozzá szükséges tanuló adathalmaz felépítése: végrehajtja az implementált algoritmust létrehozva a megadott nevű output fájlokat. Választhatunk a két bemutatott lehetőség közül a negatív példák meghatározására.

V: frameResults

7. Szöveg megjelenítése. (Bemutatás, állapot, eredmények)

Lekérdezés újraírása

Ha kész a döntési fa, akkor az újraírt feltételeket megkapjuk a gyökértől a pozitív levelekig vett utak diszjunkciójával („VAGY” művelet). Az úton lévő feltételek „ÉS” kapcsolatban állnak (konjunkció). Például:



A és C vagy nem A és nem B

Értelemszerűen az attribútum lista változatlan, a tábla kifejezésben legalább azokat a táblákat (és a hozzájuk tartozó fix feltételt, ha van) hagyjuk meg, amiknek oszlopai szerepelnek az attribútumok vagy a feltételek közt.

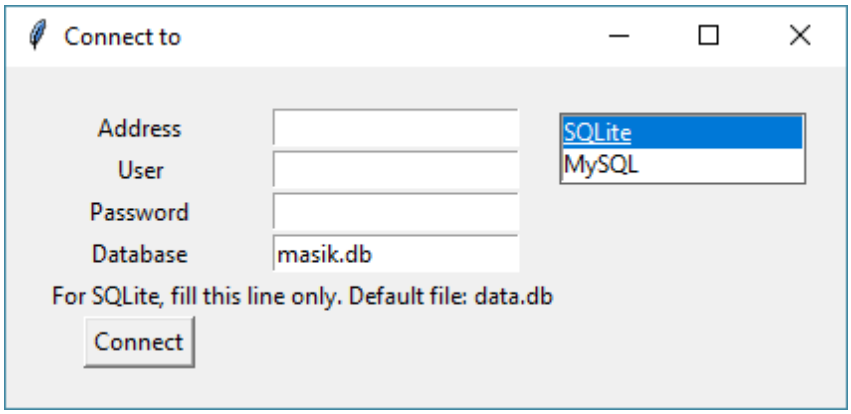
Tesztelés

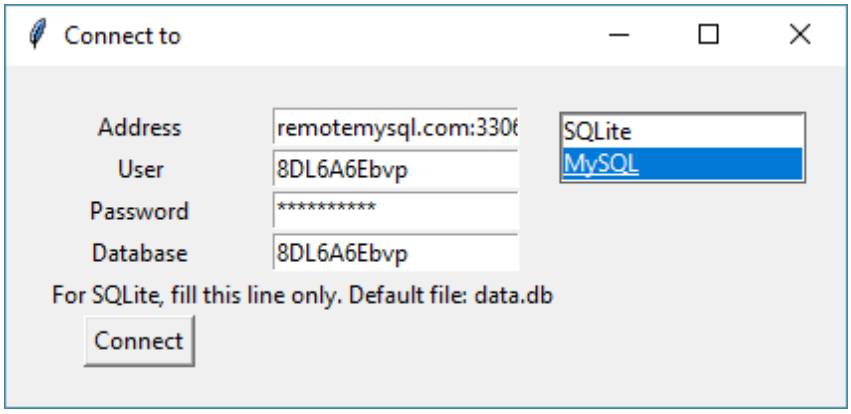
Az algoritmus hatékonyságáról részletes elemzés olvasható a publikációban. [1] Lényegében a kritikus része a lekérdezés negáltjának pontossága. A heurisztikus módszer kevés feltételnél pontatlan, átlagosan az adatbázis 20-30%-os arányában, maximum 60-80%. 4 feltételtől az átlag 10% alatt, de még mindig vannak kiugró esetek. 6-tól lesz igazán pontos, az átlag és a max. is 0% (nagyon pici eltérés).

Próbáljuk ki a felhasználói felület funkcióit!

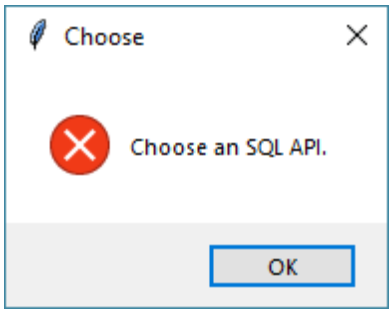
1. a) Indításkor csatlakozik az adatfájlhoz, megjelenik a táblák listája.

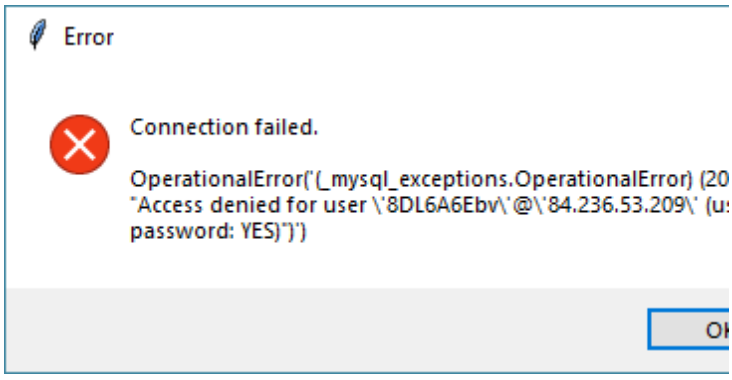
b) Csatlakozhatunk másik fájlhoz és MySQL adatbázishoz is.



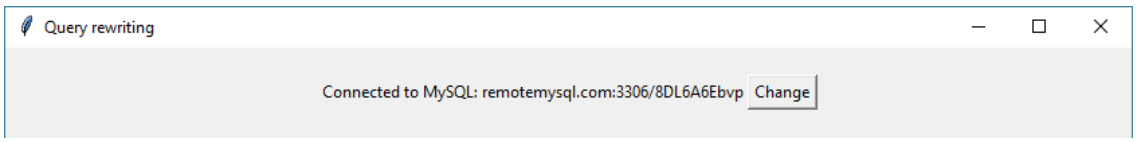


Ha nem választunk a listából vagy sikertelen a csatlakozás, hiba üzenetet kapunk és nem bontja a már élő kapcsolatot.

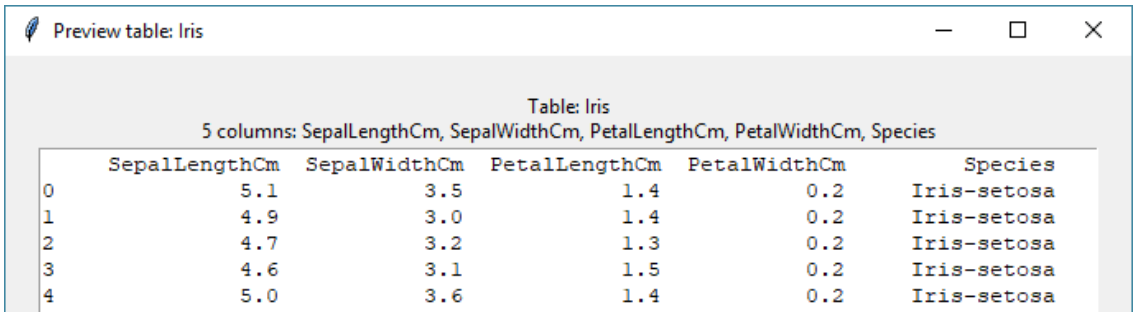


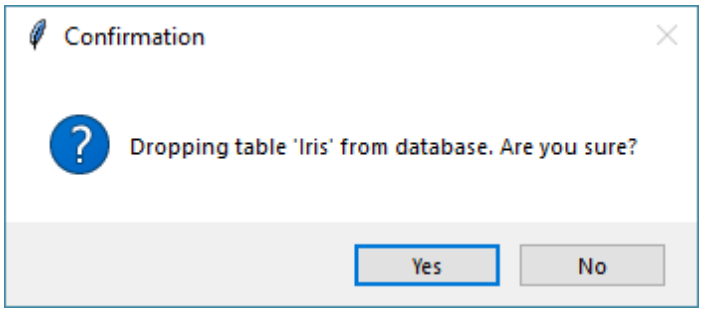


Ha sikerül csatlakozni, a megerősítést a felső állapotmezőben látjuk.



2. Frissül a táblák listája, ahol megtekinthetjük a táblák első és utolsó 30 sorát (Pandas.DataFrame alapján), illetve törölhetjük őket egy megerősítés után.

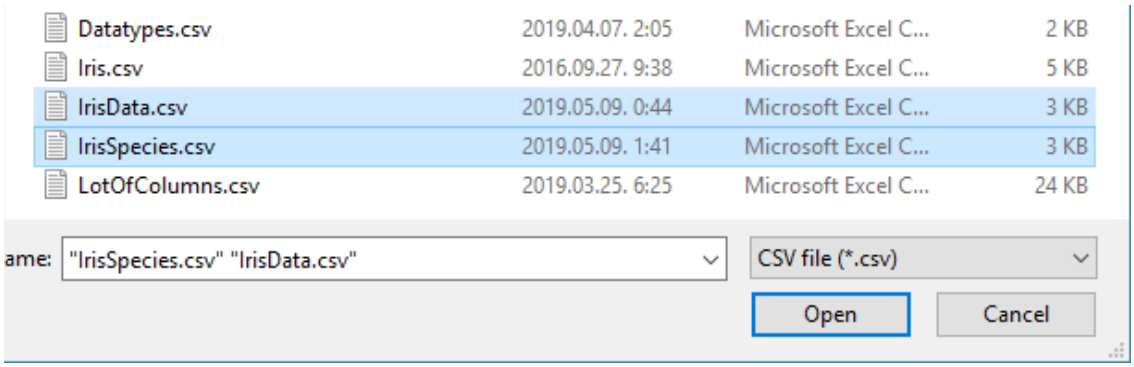




Törlés után eltűnik az adott tábla sora.

3. Betölthetünk tetszőleges számú CSV fájlt. A fájl neve lesz a tábla neve, az első sora a tábla oszlopainak neve. Az első oszlop eldobható (hasznos sorszámozás esetén). Megadható egy vagy több index is, az oszlop sorszámával 0-tól kezdődően, szóközzel vagy vesszővel elválasztva.

```
Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
```



Az új táblák sorai megjelennek és helyesen működnek.

4. A lekérdezés szövege nincs ellenőrizve, a felhasználóra bízzuk, hogy olyat adjon meg, ami értelmes, illetve hasznos eredményt hoz. Figyelni kell az SQL nyelv használatára is, pl. a Datatypes táblában kulcsszavakat használunk az oszlopok nevének.

```
Id,Int,Float,String,Datetime,Bool
1,1,3.5,Iris-setosa,2019-04-06 21:07:29.101078,True
```

Ez esetben egy lekérdezés pl. SELECT `Int`, `Float` FROM Datatypes WHERE `Int` > 0

Vagy táblák összekapcsolásakor az azonos nevű oszlopokat és táblákat átnevezni:

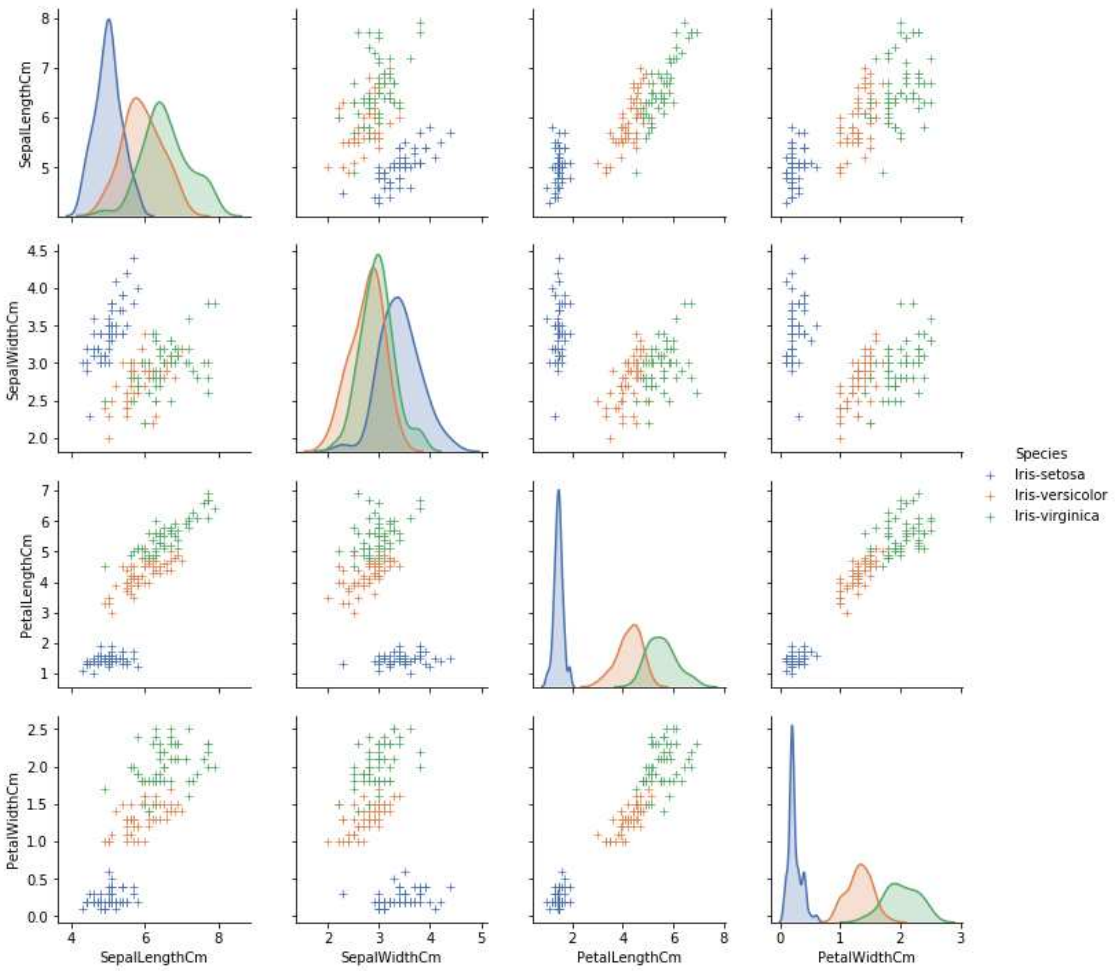
Select I1.Species, I2.Species from Iris I1, Iris I2 where 1

Érdemes az algoritmus futtatása előtt ellenőrizni a lekérdezést, ezért került be ez a funkció.

5. Kiírja az optimális negáltat és a méretét. Ha több egyező méretű van, akkor mindet. Az algoritmus ezek közül az elsőt használja fel. A heurisztikus opció csak egyet ad vissza.

6. Az algoritmust a már említett „Iris” adathalmazzal teszteljük. Írisz virágok osztályzása 150 sorban 3 fajra szirom- (petal) és csészelevelek (sepal) mérete alapján egyenlő eloszlásban (50-50-50). Egy részlet és az adathalmaz ábrázolása:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
10	4.9	3.1	1.5	0.1	Iris-setosa
20	5.1	3.8	1.5	0.3	Iris-setosa
30	4.7	3.2	1.6	0.2	Iris-setosa
40	5.1	3.4	1.5	0.2	Iris-setosa
50	5.0	3.3	1.4	0.2	Iris-setosa
60	5.2	2.7	3.9	1.4	Iris-versicolor
70	5.6	2.5	3.9	1.1	Iris-versicolor
80	5.7	2.6	3.5	1.0	Iris-versicolor
90	5.5	2.5	4.0	1.3	Iris-versicolor
100	5.7	2.8	4.1	1.3	Iris-versicolor
110	7.2	3.6	6.1	2.5	Iris-virginica
120	6.0	2.2	5.0	1.5	Iris-virginica
130	7.2	3.0	5.8	1.6	Iris-virginica
140	6.9	3.1	5.4	2.1	Iris-virginica
150	5.9	3.0	5.1	1.8	Iris-virginica



Ez a halmaz népszerű osztályzások tesztelésére, mert kis mérete miatt könnyen ellenőrizhető és az ábrán láthatóan van benne teljesen különálló csoport és olyanok is, amiknek egy kisebb része összeér. Így széles skáláját kapjuk a pontok osztályzásának nehézségére.

Először próbáljuk meg a különálló halmazt kiválasztani!

> Select Species from Iris where SepalLengthCm < 5.9 and PetalLengthCm < 2 and PetalWidthCm < 0.7

Körül határoltam 3 értékét, de a grafikonokon jól látható, hogy az egyik szirom méret is elég lenne. Lássuk a döntési fát!

> A program, vagyis Scikit-learn default algoritmusa alapján: Species = Iris-setosa
>
> C5.0 alapján: PetalWidthCm <= 0.8

Felismerte nem csak az egyszerűbb, rövidebb feltételt, de az adatok mögötti jelentést is.

Most nézzünk egy másik fajt és próbáljuk ki fordítva!

Select * from Iris where Species = ”Iris-versicolor”

Ki tudja-e választani számunkra a faj megkülönböztető értékeit? A fenti ábrával megint könnyű dolgunk van, mert nem mindegyiken kavarodnak nagyon össze. Legkevésbé talán a PetalLength – PetalWidth grafikonokon.

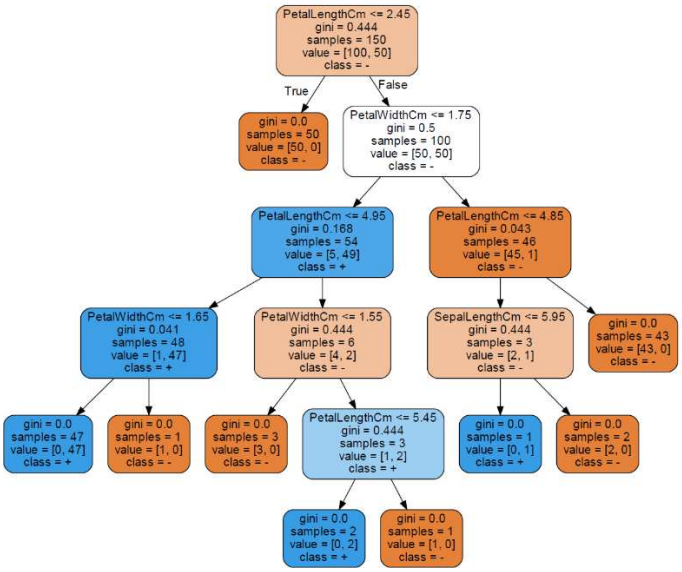
A program elég nagy fát épít, 8 feltétel van benne. Összevonom, amiket lehet, hogy rövidebb legyen:

PetalWidthCm > 0.8 and PetalWidthCm <= 1.65 and PetalLengthCm <= 4.95 OR PetalWidthCm > 1.55 and PetalWidthCm <= 1.75 and PetalLengthCm > 4.95 and SepalLengthCm <= 6.95 OR
PetalWidthCm > 1.75 and PetalLengthCm <= 4.85 and SepalWidthCm > 3.1

A három feltételnek rendre 47, 2 és 1 darab adatsor felel meg. Ezt használja fel a C5.0, hogy hatékonyabb legyen. A másik 3 különálló adatsort elhagyja a végeredményből

PetalLengthCm > 2.4 and PetalLengthCm <= 4.9 and PetalWidthCm <= 1.7

Ez sokkal látványosabb eredmény, pontosan azt a két attribútum alapján meghatározott területet kaptuk, amit a grafikon alapján becsültünk. Ha futtatjuk az új lekérdezést, 48 sort kapunk. Bár hármat elhagytunk a halmazból, egy újat felfedeztünk.




```
Decision tree:

PetalLengthCm <= 2.4: - (50)
PetalLengthCm > 2.4:
:...PetalWidthCm > 1.7: - (46/1)
    PetalWidthCm <= 1.7:
    :...PetalLengthCm <= 4.9: + (48/1)
        PetalLengthCm > 4.9: - (6/2)


Evaluation on training data (150 cases):

          Decision Tree
        ----------------
        Size      Errors

           4    6( 4.0%)   <<


         (a)   (b)    <-classified as
        ----  ----
          47     3    (a): class +
           3    97    (b): class -
```

Próbáljuk ki a táblák csatolását és a fix feltételt! Ehhez különvetten a Species oszlopot egy másik táblába. Állítsuk a különböző virágokat párba! A fix feltételek most nagyon hasznosak, kizárjuk a nem különböző- és az ismételt párosításokat.

```
Select * from (IrisData join IrisSpecies on IrisData.Id = IrisSpecies.Id) as i1, Iris i2 where i1.Species != i2.Species and (i1.Species = 'Iris-setosa' or i1.Species = 'Iris-versicolor' and i2.Species='Iris-virginica')
```

Az első aliast sajnos nem használhatjuk. A tanuló halmaz felépítéséhez szükség van az attribútum listára, amit nem ismerünk és a „*”-gal érünk el. Összekapcsolt táblák esetén táblánként kell ezt tennünk (i1.*, i2.*) az azonos oszlopnevek miatt. Viszont „i1.*”-ot nem ismeri fel.

```
Select * from IrisData join IrisSpecies on IrisData.Id = IrisSpecies.Id, Iris i2 where IrisSpecies.Species != i2.Species and (IrisSpecies.Species = 'Iris-setosa' or IrisSpecies.Species = 'Iris-versicolor' and i2.Species='Iris-virginica')
```

Egy kicsit körülményesebb, de így már rendben van. A következő probléma, hogy az „Id” mezők is bekerültek a táblába és így a döntési fába is, ahol haszontalan zajként vannak jelen. A megoldás, hogy a táblák kifejezésében és a fix feltételek közt szereplő oszlopokat kihagyjuk.

Legyen a keresési feltétel, hogy az általában nagyobb virág (2. táblában foglal helyet) két hossz mérete kisebb a másikénál.

```
IrisData.SepalLengthCm > i2.SepalLengthCm and IrisData.PetalLengthCm > i2.PetalLengthCm
```

Azt kaptuk, hogy IrisData.PetalLengthCm <= 5, de három sor nem esik bele. Igazából 5.1-nek kéne lennie, és akkor minden sorra jó. Hiába, még a legjobb módszerben is van pontatlanság. Ez a feltétel egyáltalán nem hasonló halmazt határoz meg, 25 sor helyett 7450, viszont egy lényeges információt megtudunk belőle. Ahhoz, hogy az általában nagyobb virág hossza kisebb legyen a másik fajnál, a sziromlevelek hosszának 5.1 cm alatt kell lennie.

Fejlesztési lehetőségek

Model

- Újraírt lekérdezés teljes szöveges meghatározása

- Zárójeles tábla kifejezésre (több táblára) alias

- Lekérdezés szabályainak gyengítése (pl. lehet benne VAGY, ez esetben a negált kifejezést a diszjunkciók tagjaira külön nézzük)

- Vagy egymásba ágyazott műveletek (ANY, ALL, EXISTS) átírása a szabályoknak megfelelő tábla kapcsolat + külső kulcsra

Controller

- Már említett többféle adatbázis API-ra bővítés (ez a modelben is kis alakítás)

- Több kapcsolat egyszerre

- Táblák vagy lekérdezések mentése CSV-be

- Táblaműveletek futtatása (Create table, alter table, insert into)

Nézet

- Alapértelmezett kapcsolat mentése

- Kedvenc kapcsolatok mentése

- Kedvenc lekérdezések mentése

- Bevitt lekérdezés ellenőrzése

Felhasználói dokumentáció

Fő funkció

Adat **felfedezés**t (data exploration) szeretnénk csinálni egy lekérdezés alapján **két okból** is. Egyrészt **mintát** vagy összefüggést keresünk a válasz sorok közt, amit felhasználhatunk az adathalmaz megértéséhez. Másrészt **újabb adatsorokat** keresünk, amelyek nagyon hasonlóak az eredeti lekérdezés válaszaihoz. Ezzel bővítve a célhalmazunk. (Gondoljunk pl. vásárlókra, termék ajánlásra, új égitestek keresésére vagy gépi tanulásra!)

Ezt a lekérdezés, pontosabban a WHERE **feltételek újraírásá**val hajtjuk végre. Először meghatározzuk a pozitív és negatív példákat, amiket szeretnénk a válasz halmazban látni, illetve nem. A két halmaz alapján felépítjük a **döntési fá**t, ami egyben meghatározza az új lekérdezést.

A döntési fa feltételei alapján szeretnénk mintákat felfedezni és az újraírt lekérdezés válaszai adják a bővebb adathalmazt.

Rendszerkövetelmény

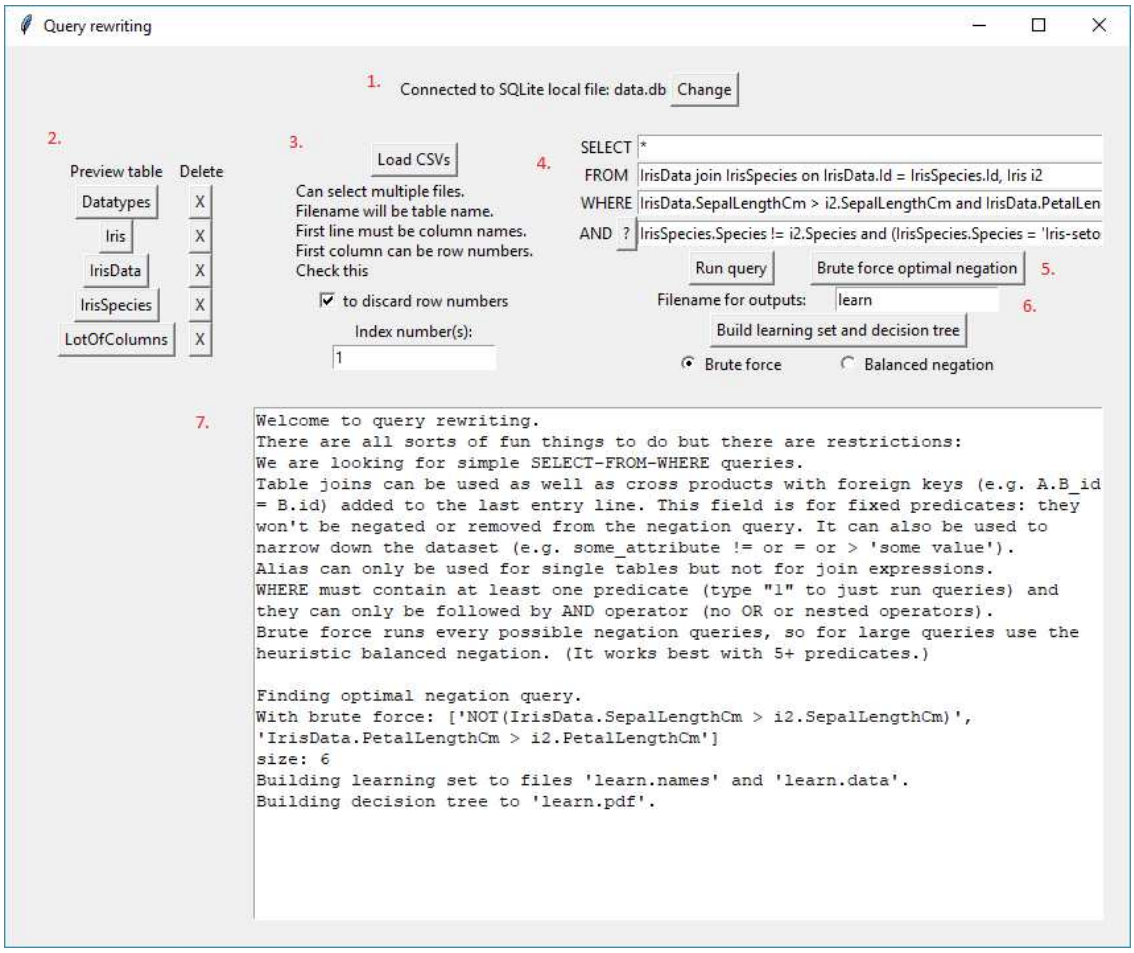
A kód Python 3, használatához szükség van Pythonra a felhasznált könyvtárakkal.

Az Anaconda programcsomagot ajánlom. Ebben benne van minden környezet, a „graphviz”, „graphviz-python”, „mysqlclient” és „sqlparse” nincsenek alapból telepítve. (Pár percet vesz igénybe az Anaconda Navigatorban.)

A program SQLite adatfájlján kívül MySQL adatbázis használható.

(Szükség esetén PyInstallerrel Windows exe fájl készíthető.)

Program használata

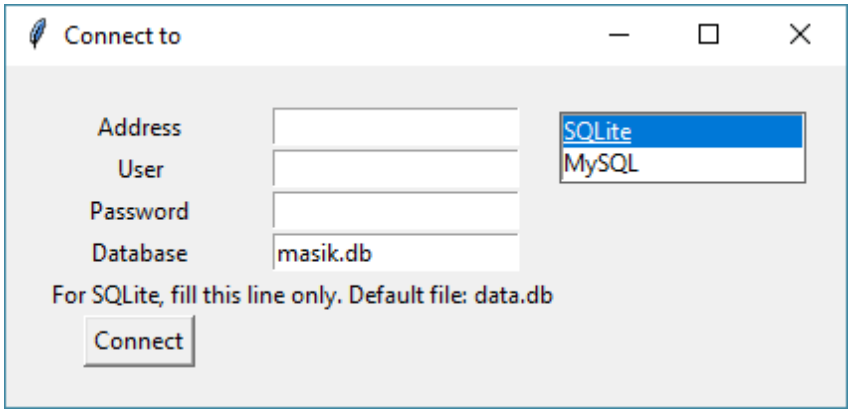


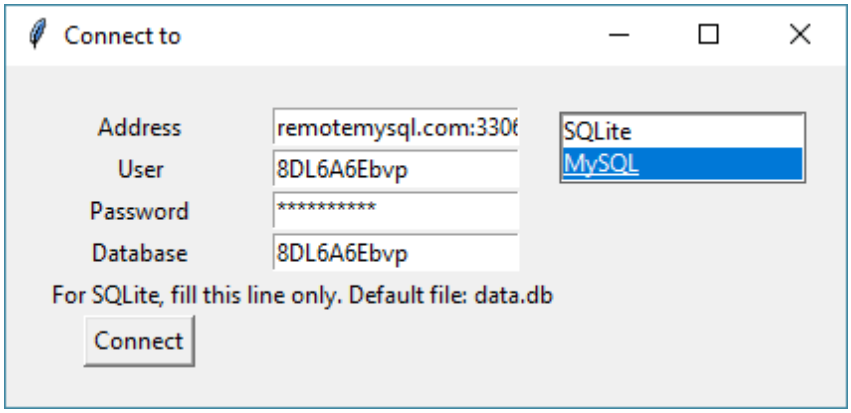
Piros számozással jelölve a funkciók.

1. Kapcsolódás adatbázishoz:

a) Indításkor a program könyvtárában létrehozott „data.db” nevű SQLite adatfájlhoz kapcsolódik. Ezt tetszés szerint átnevezhetjük és újakat hozhatunk létre.

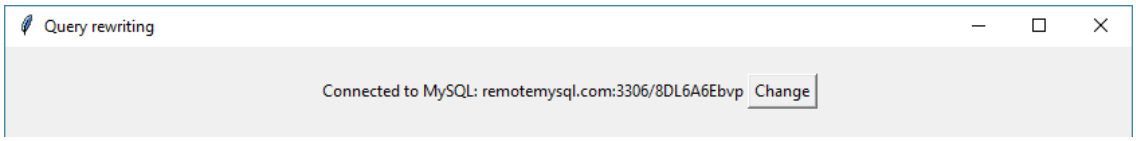
b) Kapcsolódhatunk másik adatfájlhoz vagy adatbázishoz. Jelenleg az SQLite és MySQL opciók elérhetők.





Ha nem választunk a listából vagy sikertelen a csatlakozás, hiba üzenetet kapunk és nem bontja a már élő kapcsolatot.

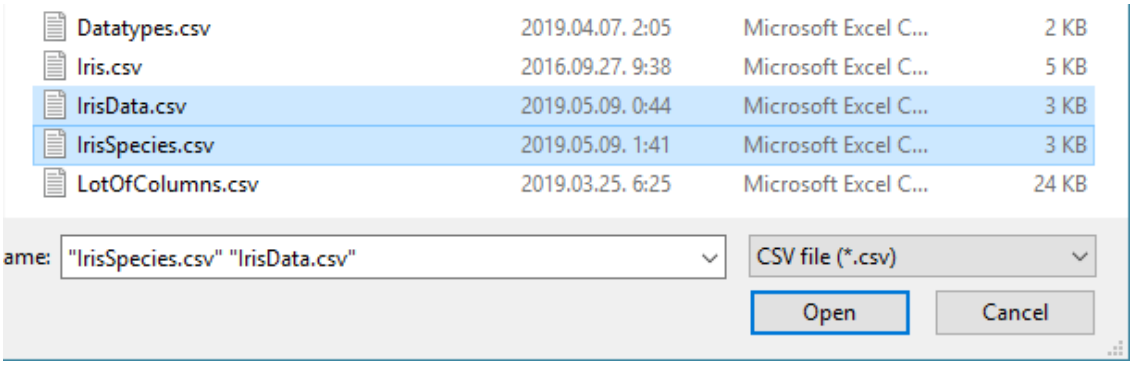
Ha sikerül csatlakozni, a megerősítést a felső állapotmezőben látjuk.



2. Táblák listája, előnézet, törlés: kapcsolódás után látjuk az adatbázis tábláinak listáját, ahonnan megnézhetjük külön ablakban az első és utolsó 30 sorát és törölhetjük a táblát.

3. Új tábla betöltése CSV fájlból: tetszőleges számú CSV fájlt tölthetünk be. Fájlnév lesz a tábla neve, az első sor az oszlopok fejléce. Az első oszlop eldobható (hasznos sorszámozás esetén). Megadható egy vagy több index is, az oszlop sorszámával 0-tól kezdődően, szóközzel vagy vesszővel elválasztva.

```
Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
```



4. Lekérdezés megadása és futtatása: külön ablakban írja ki a választ.

A lekérdezésre vonatkozó megszorítások a program indításakor olvashatóak. Join kifejezések megadhatók, de aliast csak egy táblára lehet használni, kifejezésre egyelőre nem. Legalább 1 feltételnek szerepelnie kell, a feltételek közt csak „ÉS” kapcsolat lehet. Az utolsó sorba fix feltételek kerülnek: ezek nem lesznek negálva vagy elhagyva az algoritmus során.

A lekérdezés szövege nincs ellenőrizve, a felhasználóra van bízva, hogy olyat adjon meg, ami értelmes, illetve hasznos eredményt hoz. Figyelni kell az SQL nyelv használatára is, pl. a Datatypes táblában kulcsszavakat használunk az oszlopok nevének.

```
Id,Int,Float,String,Datetime,Bool
1,1,3.5,Iris-setosa,2019-04-06 21:07:29.101078,True
```

Ez esetben egy lekérdezés pl. SELECT `Int`, `Float` FROM Datatypes WHERE `Int` > 0

Vagy táblák összekapcsolásakor az azonos nevű oszlopokat és táblákat átnevezni:

Select I1.Species, I2.Species from Iris I1, Iris I2

Érdemes az algoritmus futtatása előtt ellenőrizni a lekérdezést, ezért került be ez a funkció.

5. Optimális negált meghatározása az összes lehetőség futtatásával.

(Negáltnak nevezzük a negatív példákat adó lekérdezést. Optimális, ha a negatív példák száma a legközelebbi a pozitívakéhoz.)

Kiírja az optimális negáltat és a méretét. Ha több egyező méretű van, akkor mindet. Az algoritmus ezek közül az elsőt használja fel. (A heurisztikus opció csak egyet ad vissza.)

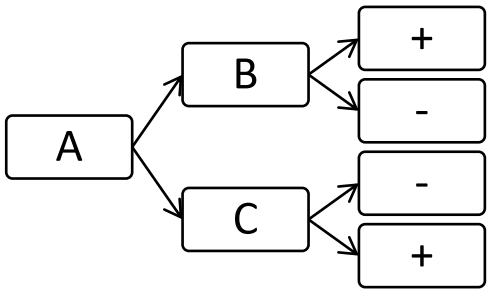
6. Döntési fa és a hozzá szükséges tanuló adathalmaz felépítése: létrehozza a megadott nevű output fájlokat. Választhatunk két lehetőség közül a negatív példák meghatározására: egyszerű és heurisztikus.

Az utóbbi legalább 6 feltételnél és nagy táblák esetén ajánlott, az összes lehetséges lekérdezés futtatása helyett becslést ad a méretükre. Kevés feltételnél pontatlan, átlagosan az adatbázis 20-30%-os arányában, maximum 60-80%. (6-tól 0% közeli.)

7. Szöveg megjelenítése: a program használatának rövid bemutatás, állapot- és hibajelzések, eredmények

Lekérdezés újraírása

Ha kész a döntési fa, akkor az újraírt feltételeket megkapjuk a gyökértől a pozitív levelekig vett utak diszjunkciójával („VAGY” művelet). Az úton lévő feltételek „ÉS” kapcsolatban állnak (konjunkció). Például:



A és C vagy nem A és nem B

Értelemszerűen az attribútum lista változatlan, a tábla kifejezésben legalább azokat a táblákat (és a hozzájuk tartozó fix feltételt, ha van) hagyjuk meg, amiknek oszlopai szerepelnek az attribútumok vagy a feltételek közt.

Referenciák

[1] Julien Cumin, Jean-Marc Petit, Vasile-Marian Scuturici, Sabina Surdu: „Data Exploration with SQL using Machine Learning Techniques” EDBT, 2017. https://openproceedings.org/2017/conf/edbt/paper-155.pdf

Hasonló témájú publikációk találhatók [1] „Related work” részében.

[2] SINTEF: „Big Data, for better or worse: 90% of world's data generated over last two years” ScienceDaily, 2013. május 22. https://www.sciencedaily.com/releases/2013/05/130522085217.htm

[3] The Large Synoptic Survey Telescope https://www.lsst.org/ Online, 2019.

[4] SKA Telescope — Frequently Asked Questions about the SKA https://www.skatelescope.org/frequently-asked-questions/ Online, 2019.

[5] Inverse — The World's First-Ever Black Hole Photo Was an Epic Feat of Data Storage, https://www.inverse.com/article/54833-m87-black-hole-photo-data-storage-feat Online, 2019

[6] OpenAI Five https://openai.com/blog/openai-five/ Online, 2019

[7] Zephoria Inc. — Top 20 Facebook Statistics https://zephoria.com/top-15-valuable-facebook-statistics/ Online, 2019

[8] scikit-learn 0.20.3 documentation — 1.10. Decision Trees https://scikit-learn.org/stable/modules/tree.html Online, 2019

[9] Anaconda https://www.anaconda.com/ Online, 2019

[10] RuleQuest Research Data Mining Tools — Free Downloads https://www.rulequest.com/download.html Online, 2019